```
// This is a list of the variables available in the EventData class and its subclasses. Here,
// "event" is an instance of the EventData class. To see the full class structure of the
// EventData class, browse the source files in the darkart/Products/ directory of the repository.

////////////////////////////////////////////////////////////////////////////////////////////
// The event level identifier information, such as event ID and event timing information, are  //
// stored in event->event_info. The event-level pulse information is stored in event->pulses   //
// and is described at the end.                                                                //
////////////////////////////////////////////////////////////////////////////////////////////

int      event->event_info.run_id
int      event->event_info.event_id
uint32_t event->event_info.gps_coarse
uint32_t event->event_info.gps_fine
uint32_t event->event_info.pps
uint32_t event->event_info.total_inhibit_time_us //total trigger inhibit time (us)
uint32_t event->event_info.incremental_inhibit_time_20ns //trigger inhibit time for the previous
                                                         //trigger (20 ns)
uint32_t event->event_info.live_time_20ns //live time for the current trigger (20 ns)
uint64_t event->event_info.timestamp_sec //unix timestamp for this event
uint64_t event->event_info.dt_usec //time since the last event in microseconds
uint64_t event->event_info.event_time_usec //time since run start in microseconds
int      event->event_info.nchans //physical channels that lit up
bool     event->event_info.saturated //true if any channel hit the limit of its digitizer


////////////////////////////////////////////////////////////////////////////////////////////
// The physical channels are accessed by event->channels[i]. The index i runs over 0 to 37 but //
// i is NOT the channel ID! To access the SUM CHANNEL, use event->sumchannel, which is of the  //
// same class type as event->channels[i]                                                       //
////////////////////////////////////////////////////////////////////////////////////////////

int     event->channels[i].channel.channel_id() //returns the unique global identifier for this channel
double  event->channels[i].channel.sample_rate //samples per microsecond
int     event->channels[i].channel.trigger_index
int     event->channels[i].channel.nsamps //number of samples in the waveform; should be the same for
                                          //all channels
bool    event->channels[i].channel.saturated //did the signal hit the max or min range of the digitizer?

double  event->channels[i].pmt.spe_mean
//the other values for the pmt object are not currently being filled

double  event->channels[i].raw_wfm.minimum
double  event->channels[i].raw_wfm.maximum
int     event->channels[i].raw_wfm.min_index
int     event->channels[i].raw_wfm.max_index
double  event->channels[i].raw_wfm.min_time
double  event->channels[i].raw_wfm.max_time
int     event->channels[i].raw_wfm.nsamps //number of samples in the waveform

double  event->channels[i].baseline_subtracted_wfm.minimum
double  event->channels[i].baseline_subtracted_wfm.maximum
//and so on; baseline_subtracted_wfm is the same class type as the raw_wfm object

double  event->channels[i].integral.minimum
double  event->channels[i].integral.maximum
//and so on; integral is the same class type as the raw_wfm object

bool    event->channels[i].baseline.found_baseline
double  event->channels[i].baseline.mean
double  event->channels[i].baseline.variance
bool    event->channels[i].baseline.saturated
int     event->channels[i].baseline.length
int     event->channels[i].baseline.search_start_index

int     event->channels[i].regions[j].region_number
double  event->channels[i].regions[j].start_time
double  event->channels[i].regions[j].end_time
int     event->channels[i].regions[j].start_index
int     event->channels[i].regions[j].end_index
double  event->channels[i].regions[j].max
double  event->channels[i].regions[j].max_time
double  event->channels[i].regions[j].min
```

```
double  event->channels[i].regions[j].min_time
double  event->channels[i].regions[j].integral

int     event->channels[i].pulses[j].pulse.pulse_id() //unique identifier for this pulse within channel
bool    event->channels[i].pulses[j].pulse.start_clean //start of pulse does not overlap with previous one
bool    event->channels[i].pulses[j].pulse.end_clean //end of pulse does not overlap with next pulse
int     event->channels[i].pulses[j].pulse.start_index
int     event->channels[i].pulses[j].pulse.end_index
double  event->channels[i].pulses[j].pulse.start_time
double  event->channels[i].pulses[j].pulse.end_time
double  event->channels[i].pulses[j].pulse.dt //time between start of this pulse and the previous one

bool    event->channels[i].pulses[j].param.found_peak //did we find a peak?
int     event->channels[i].pulses[j].param.peak_index
double  event->channels[i].pulses[j].param.peak_time
double  event->channels[i].pulses[j].param.peak_amplitude
double  event->channels[i].pulses[j].param.integral
bool    event->channels[i].pulses[j].param.peak_saturated
std::vector<double> event->channels[i].pulses[j].param.f_param //f-parameters for different time
                                                               //values
double  event->channels[i].pulses[j].param.f90 //f-parameter for 90 ns
double  event->channels[i].pulses[j].param.t05 //time to reach XX% of total integral
double  event->channels[i].pulses[j].param.t10
double  event->channels[i].pulses[j].param.t90
double  event->channels[i].pulses[j].param.t95
double  event->channels[i].pulses[j].param.fixed_int1 //integral of first 7 us of the pulse
double  event->channels[i].pulses[j].param.fixed_int2 //integral of first 30 us of the pulse
bool    event->channels[i].pulses[j].param.fixed_int1_valid //did the event extend past the
                                                            //integration window?
bool    event->channels[i].pulses[j].param.fixed_int2_valid
double  event->channels[i].pulses[j].param.npe //integral scaled for single pe amplitude

////////////////////////////////////////////////////////////////////////////////////////////
// The event-level pulse information is stored in PulseData objects that are the same as the  //
// event->channels[i].pulses[i]. These event-level variables are, for the most part, built by  //
// totalling the corresponding pulse parameters across the physical channels.                  //
////////////////////////////////////////////////////////////////////////////////////////////

int     event->pulses[j].pulse.pulse_id() //unique identifier for this pulse
bool    event->pulses[j].pulse.start_clean //start of pulse does not overlap with previous one
bool    event->pulses[j].pulse.end_clean //end of pulse does not overlap with next pulse
int     event->pulses[j].pulse.start_index
int     event->pulses[j].pulse.end_index
double  event->pulses[j].pulse.start_time
double  event->pulses[j].pulse.end_time
double  event->pulses[j].pulse.dt //time between start of this pulse and the previous one

bool    event->pulses[j].param.found_peak //did we find a peak?
int     event->pulses[j].param.peak_index
double  event->pulses[j].param.peak_time
double  event->pulses[j].param.peak_amplitude
double  event->pulses[j].param.integral
bool    event->pulses[j].param.peak_saturated
std::vector<double> event->pulses[j].param.f_param //f-parameters for different time values
double  event->pulses[j].param.f90 //f-parameter for 90 ns
double  event->pulses[j].param.t05 //time to reach XX% of total integral
double  event->pulses[j].param.t10
double  event->pulses[j].param.t90
double  event->pulses[j].param.t95
double  event->pulses[j].param.fixed_int1 //integral of first 7 us of the pulse; inverted and
                                          //scaled to be in units of p.e.
double  event->pulses[j].param.fixed_int2 //integral of first 30 us of the pulse; inverted and
                                          //scaled to be in units of p.e.
bool    event->pulses[j].param.fixed_int1_valid //did the event extend past the integration window?
bool    event->pulses[j].param.fixed_int2_valid
double  event->pulses[j].param.npe //integral scaled for single pe amplitude
```